Bike sharing demand

작성일 : 2023-10-20

학번 : 2023254002

성명 : 이민수

목 차

데이터 전 처리 및 시각화

1. 큰 그림 보기

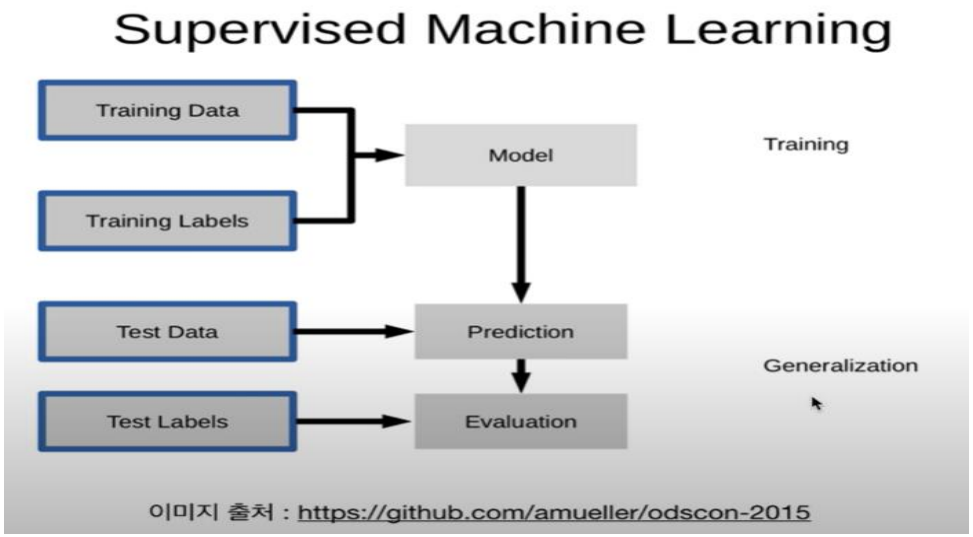
1.1 문제 정의

- **모델의 목표와 최종적인 목적**

워싱턴 D.C.의 Capital Bikeshare 프로그램에서 자전거 대여 수요를 예측하기 위해 과거 사용 패턴과 날씨 데이터를 결합하여 수요를 예측한다.

- **모델 시스템 설계**

레이블이 있고 특정 상황(Count)에 대한 값이 있기 때문에 지도 학습이며, 분류와 회귀중 자전거 대여량을 예측하는 문제이기 때문에 회귀에 속한다. Train data를 사용하여 Test data의 Count 값을 예측한다.



데이터 전 처리 및 시각화

1. 큰 그림 보기

1.2. 성능 측정 지표 선택

- **RMSLE**

과대 평가 된 항목보다는 과소평가 된 항목에 패널티를 준다.

오차(Error)를 제곱(Square)해서 평균(Mean)한 값의 제곱근(Root) 으로 값이 작을 수록 정밀도가 높다.

0에 가까운 값이 나올 수록 정밀도가 높은 값이다.

Submissions are evaluated one the Root Mean Squared Logarithmic Error (RMSLE). The RMSLE is calculated as

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(p_i+1)-\log(a_i+1))^2}$$

데이터 전 처리 및 시각화

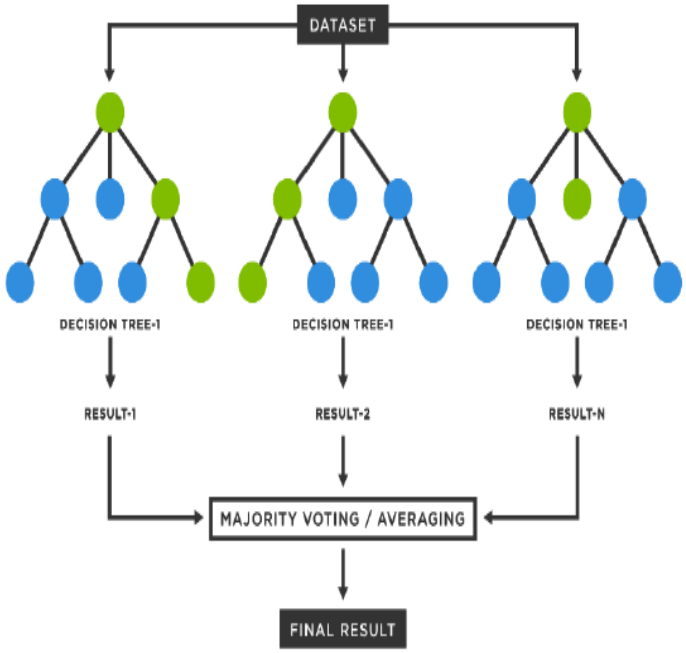
1. 큰 그림 보기

1.2. 성능 측정 지표 선택

RandomForestRegressor 이란

- 회귀 문제를 해결하기 위한 머신러닝 모델 중 하나로, 다수의 결정 트리(Decision Tree)를 앙상블하여 사용하는 모델
- 회귀 문제에서 안정적이고 강력한 성능을 보이며, 데이터셋의 특성과 크기에 상관없이 일반적으로 잘 작동하고 과적합에 강하고, 결정 트리의 단점을 보완하여 더 나은 예측 성능을 제공

- 랜덤 포레스트는 여러 개의 결정 트리를 생성하는 모델. 각 결정 트리는 데이터의 일부를 무작위로 선택하고 이 데이터로 학습. 이로 인해 각 트리는 다양한 특성과 데이터 샘플로 학습.
- 각 결정 트리를 학습할 때, 무작위로 데이터 샘플을 선택하여 사용. 이를 부트스트랩 샘플링(Bootstrap Sampling)이라고 하며, 중복을 허용한 랜덤 샘플링으로 각 트리는 서로 다른 데이터 샘플로 구성되어 다양성을 가진다.
- 각 노드에서 무작위로 선택한 특성들 중에서 최적의 분할을 찾아 각 트리가 다양한 특성을 고려하여 학습하도록 지원.
- 각 결정 트리가 예측한 결과를 모아서 최종 예측을 수행. 회귀 문제에서는 각 트리의 예측값을 평균하여 최종 예측.
- 랜덤 포레스트는 각 결정 트리의 예측을 평균화하여 예측 오차를 줄이고 모델의 일반화 성능을 향상. 이러한 앙상블 평가는 과적합을 방지하고 모델의 안정성을 높인다.
- 랜덤 포레스트는 각 특성의 중요도를 측정할 수 있다. 중요한 특성은 예측에 더 큰 영향을 미치므로, 모델의 특성 선택에 유용한 정보를 제공.
- 랜덤 포레스트는 다양한 하이퍼파라미터를 가지고 있으며, 이를 조정하여 모델의 성능을 최적화할 수 있다. 일반적으로 트리의 개수, 노드 분할 기준, 트리의 최대 깊이 등이 조정 대상이다.



데이터 전 처리 및 시각화

2. 데이터 수집

2.1 작업환경 만들기

- **데이터 수집 및 저장**

https://www.kaggle.com/c/bike-sharing-demand/ 의 Data의 항목에서 train.csv / test.csv 를 저장 및 관리

- **분석 도구 및 라이브러리 설치**

Pandas, numpy, scipy를 사용

- **시각화 도구 설정**

matplotlib, seaborn를 사용

데이터 전 처리 및 시각화

2. 데이터 수집

2.2 데이터 다운 및 구조 훑어보기

- **데이터 다운**

```
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats

# 노트북 안에 그래프를 그리기 위해
%matplotlib inline

# 그래프에서 격자로 숫자 범위가 눈에 잘 띄도록 ggplot 스타일을 사용
plt.style.use('ggplot')
```

```
# train / test
train = pd.read_csv("data/train.csv")
test = pd.read_csv("data/test.csv")
```

데이터 전 처리 및 시각화

2. 데이터 수집

2.2 데이터 다운 및 구조 훑어보기

- 구조 훑어보기

```
train.columns
```

```
Index(['datetime', 'season', 'holiday', 'workingday', 'weather', 'temp',
       'atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count'],
      dtype='object')
```

```
test.columns
```

```
Index(['datetime', 'season', 'holiday', 'workingday', 'weather', 'temp',
       'atemp', 'humidity', 'windspeed'],
      dtype='object')
```

Columns 명	데이터 내용
Datetime	시간 (YYYY-MM-DD 00:00:00)
Season	봄(1) 여름(2) 가을(3) 겨울(4)
Holiday	공휴일(1) 그외(0)
Workingday	근무일(1) 그외(0)
Weather	아주깨끗한날씨(1) 약간의 안개와 구름(2) 약간의 눈,비(3) 아주많은비와 우박(4)
Temp	온도(섭씨로 주어짐)
Atemp	체감온도(섭씨로 주어짐)
Humidity	습도
Windspeed	풍속
Casual	비회원의 자전거 대여량
Registered	회원의 자전거 대여량
Count	총 자전거 대여량 (비회원+회원)

train과 test의 컬럼에 차이가 있다. Train 데이터엔 casual, registered, count 변수가 있지만, test 변수에는 없다. 따라서 우리가 예측해야할 변수는 count라고 알 수 있다. 옆에 데이터 내용을 보면 count 변수 = casual + registered 라고 명시되어 있다.

데이터 전 처리 및 시각화

2. 데이터 수집

2.2 데이터 다운 및 구조 훑어보기

- 구조 훑어보기

```
train.head()
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10	13
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1	1

```
test.head()
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed
0	2011-01-20 00:00:00	1	0	1	1	10.66	11.365	56	26.0027
1	2011-01-20 01:00:00	1	0	1	1	10.66	13.635	56	0.0000
2	2011-01-20 02:00:00	1	0	1	1	10.66	13.635	56	0.0000
3	2011-01-20 03:00:00	1	0	1	1	10.66	12.880	56	11.0014
4	2011-01-20 04:00:00	1	0	1	1	10.66	12.880	56	11.0014

데이터 전 처리 및 시각화

2. 데이터 수집

2.2 데이터 다운 및 구조 훑어보기

- 구조 훑어보기

In [19]:
```
train['datetime'] = pd.to_datetime(train['datetime'])
test['datetime'] = pd.to_datetime(test['datetime'])
```

In [21]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   datetime    10886 non-null  datetime64[ns]
 1   season      10886 non-null  int64
 2   holiday     10886 non-null  int64
 3   workingday  10886 non-null  int64
 4   weather     10886 non-null  int64
 5   temp        10886 non-null  float64
 6   atemp       10886 non-null  float64
 7   humidity    10886 non-null  int64
 8   windspeed   10886 non-null  float64
 9   casual      10886 non-null  int64
 10  registered  10886 non-null  int64
 11  count       10886 non-null  int64
dtypes: datetime64[ns](1), float64(3), int64(8)
memory usage: 1020.7 KB
```

In [20]:
```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6493 entries, 0 to 6492
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   datetime    6493 non-null   datetime64[ns]
 1   season      6493 non-null   int64
 2   holiday     6493 non-null   int64
 3   workingday  6493 non-null   int64
 4   weather     6493 non-null   int64
 5   temp        6493 non-null   float64
 6   atemp       6493 non-null   float64
 7   humidity    6493 non-null   int64
 8   windspeed   6493 non-null   float64
dtypes: datetime64[ns](1), float64(3), int64(5)
memory usage: 456.7 KB
```

데이터의 datetime을 날짜로 인식해주기 위해서 한가지 과정을 거치었다. Pandas의 to datetime을 이용하여 datetime 컬럼을 날짜로 인식하게끔 형태를 바꾸었고, if()를 출력해보니, datetime 형태로 변경되었으며, 나머지 변수는 전부 정수 혹은 float 형태로 구성되어있는 것을 볼 수있다.

데이터 전 처리 및 시각화

2. 데이터 수집

2.2 데이터 다운 및 구조 훑어보기

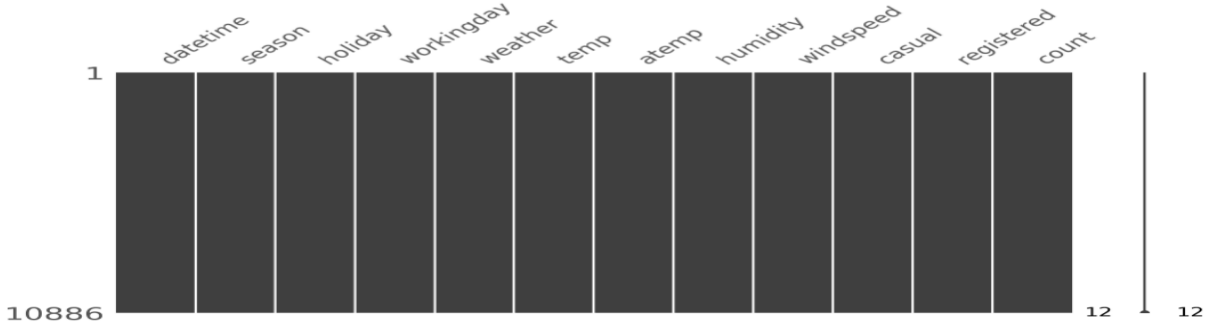
- 구조 훑어보기

```
# null 값 확인
train.isnull().sum()
```

```
datetime      0
season        0
holiday       0
workingday    0
weather       0
temp          0
atemp         0
humidity      0
windspeed     0
casual        0
registered    0
count         0
dtype: int64
```

```
# null 값을 그래프로 확인
import missingno as msno

msno.matrix(train, figsize=(12,5))
```

```
<Axes: >
```



null 값은 모두 0이며, nulle 값을 그래프로도 문제가 없음을 확인하였다.

```
In [31]: train["year"] = train["datetime"].dt.year
         train["month"] = train["datetime"].dt.month
         train["day"] = train["datetime"].dt.day
         train["hour"] = train["datetime"].dt.hour
         train["minute"] = train["datetime"].dt.minute
         train["second"] = train["datetime"].dt.second
         train.shape

Out[31]: (10886, 18)
```

데이터 시각화를 하기 위하여 datetime 형식을 연,월,일,시,분,초로 나누어 주었다.

데이터 전 처리 및 시각화

3. 데이터 이해를 위한 탐색과 시각화

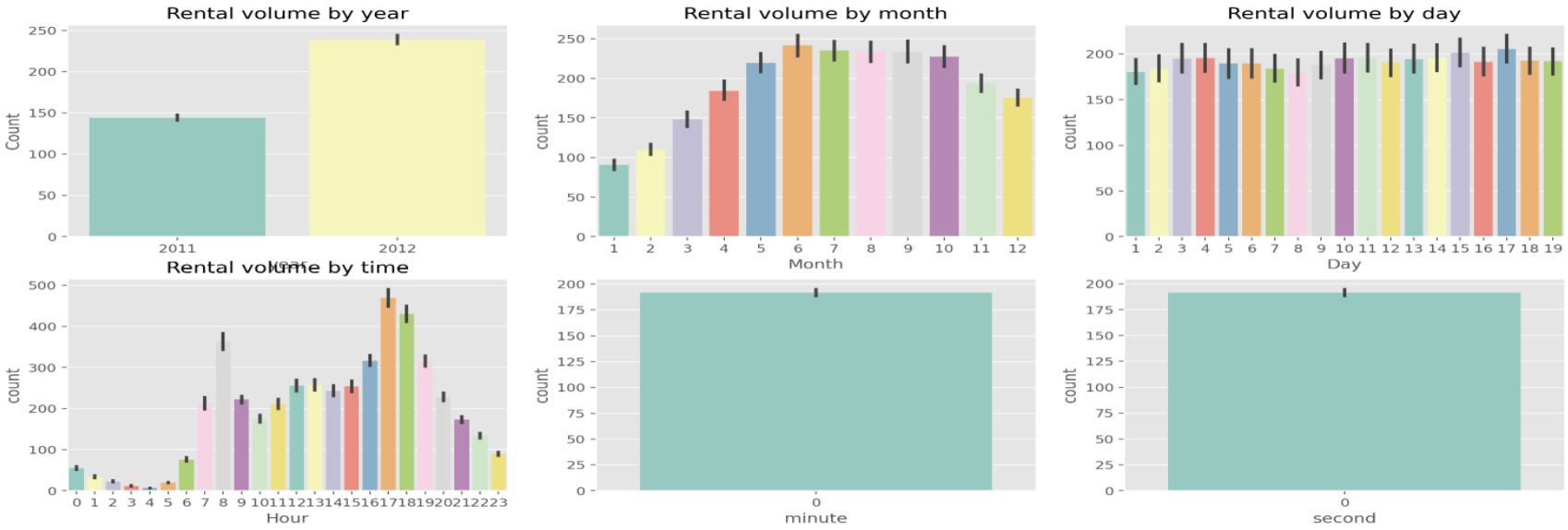
3.1 Matplotlib

In [83]:
```
year_palette = sns.color_palette("Set3", n_colors=len(train['year'].unique()))
month_palette = sns.color_palette("Set3", n_colors=len(train['month'].unique()))
day_palette = sns.color_palette("Set3", n_colors=len(train['day'].unique()))
hour_palette = sns.color_palette("Set3", n_colors=len(train['hour'].unique()))
minute_palette = sns.color_palette("Set3", n_colors=len(train['minute'].unique()))
second_palette = sns.color_palette("Set3", n_colors=len(train['second'].unique()))

figure, ((ax1, ax2, ax3), (ax4, ax5, ax6)) = plt.subplots(nrows=2, ncols=3)
figure.set_size_inches(18, 8)

sns.barplot(data=train, x="year", y="count", hue="year", ax=ax1, palette=year_palette, legend=False)
sns.barplot(data=train, x="month", y="count", hue="month", ax=ax2, palette=month_palette, legend=False)
sns.barplot(data=train, x="day", y="count", hue="day", ax=ax3, palette=day_palette, legend=False)
sns.barplot(data=train, x="hour", y="count", hue="hour", ax=ax4, palette=hour_palette, legend=False)
sns.barplot(data=train, x="minute", y="count", hue="minute", ax=ax5, palette=minute_palette, legend=False)
sns.barplot(data=train, x="second", y="count", hue="second", ax=ax6, palette=second_palette, legend=False)

ax1.set(ylabel='Count', title="Rental volume by year")
ax2.set(xlabel='Month', title="Rental volume by month")
ax3.set(xlabel='Day', title="Rental volume by day")
ax4.set(xlabel='Hour', title="Rental volume by time")
```

데이터 전 처리 및 시각화

3. 데이터 이해를 위한 탐색과 시각화

3.1 Matplotlib

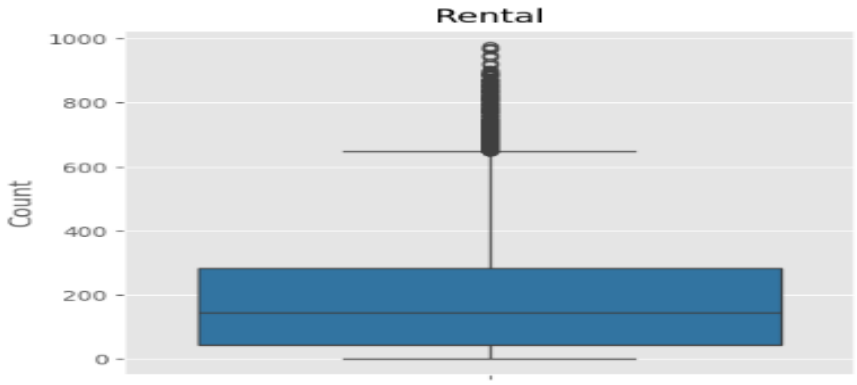


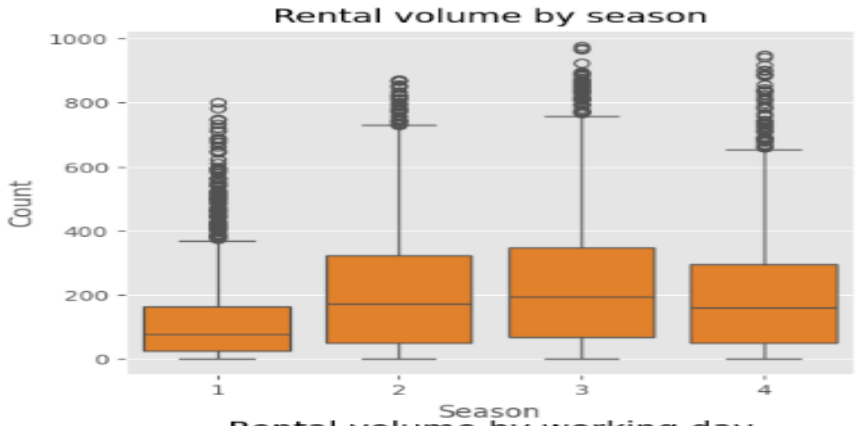
- **Rental volume by year은 2011년 보다 2012년이 더 많다.**
- **Rental volume by month는 6월이 가장 많고 7~10월이 그 뒤를 이은다. 그리고 1월이 가장 적다.**
- **Rental volume by day는 1일부터 19일까지만 있고 나머지 날짜는 test.csv에 있다. 그래서 이 데이터는 피처로 사용해서는 안 된다.**
- **Rental volume by time을 보면 출퇴근 시간에 대여량이 많은 것 같다. 하지만 주말과 나누어 볼 필요가 있을 것 같다.**
- **분, 초는 다 0이기 때문에 의미가 없다.**

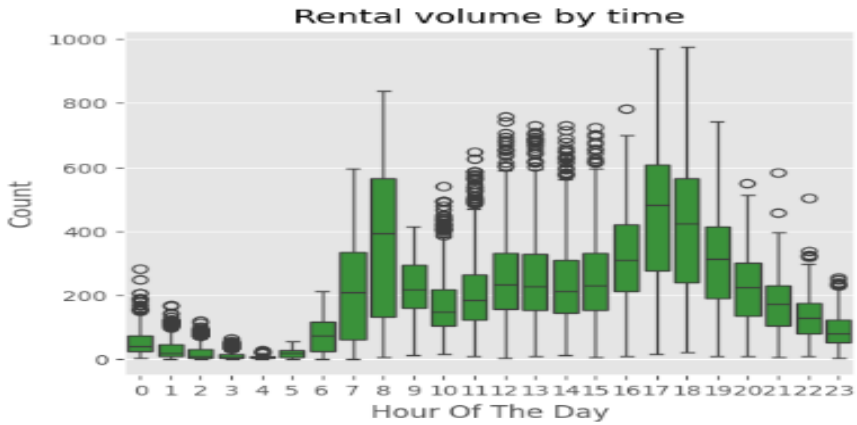
데이터 전 처리 및 시각화

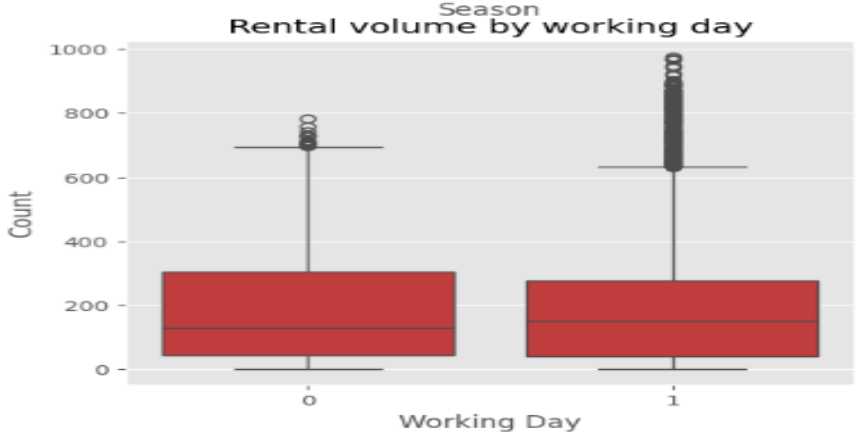
3. 데이터 이해를 위한 탐색과 시각화

3.1 Matplotlib









- **Rental volume by season에서 3분기 > 2분기 > 4분기 > 1분기 순으로 대여량이 많다.**
- **Rental volume by time은 출·퇴근 시간대에 가장 많으며 12~15시 사이가 그 뒤를 따른다.**
- **Rental volume by working day는 크게 차이가 있지 않다.**

데이터 전 처리 및 시각화

3. 데이터 이해를 위한 탐색과 시각화

3.1 Matplotlib

In [67]:
```
train["dayofweek"] = train["datetime"].dt.dayofweek
train.shape
```

Out[67]:
```
(10886, 19)
```

In [68]:
```
train["dayofweek"].value_counts()
```

Out[68]:
```
dayofweek
5    1584
6    1579
3    1553
0    1551
2    1551
1    1539
4    1529
Name: count, dtype: int64
```

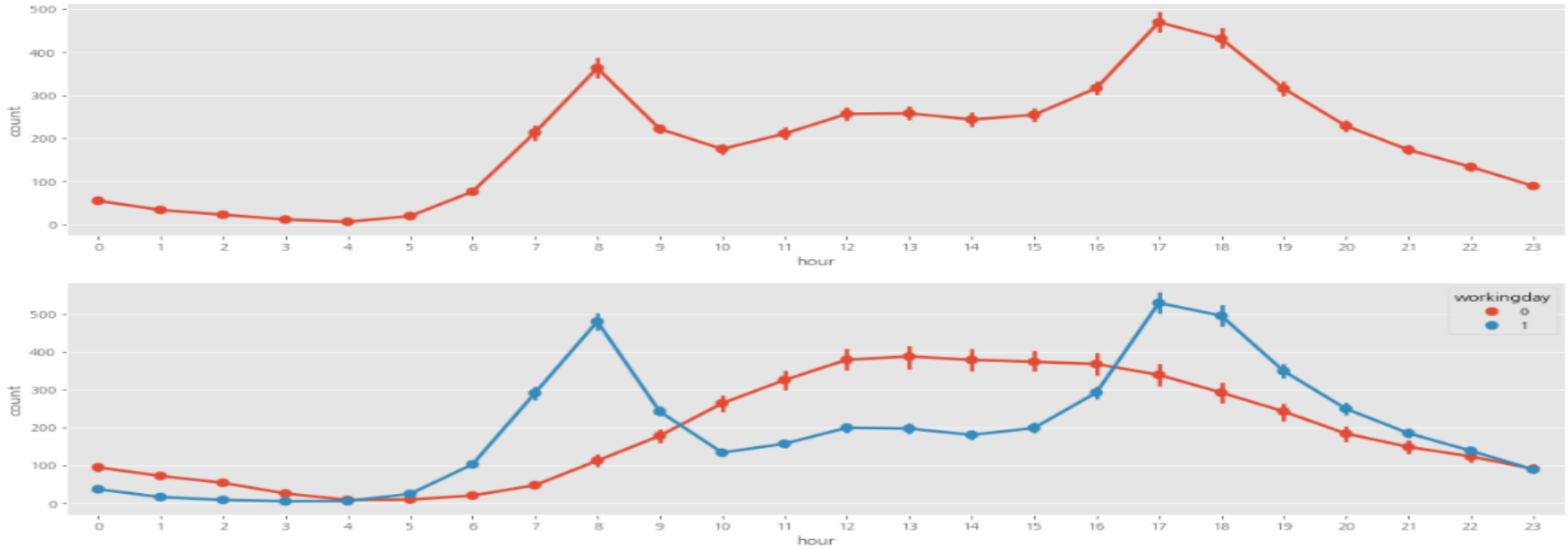
dayofweek를 사용하여 월(0) ~ 일(6)까지의 대여량을 확인하였지만, 가장 많은 대여량을 빌린 토요일 1584 와 가장 적게 대여한 금요일 1529의 차이는 55 밖에 차이가 나지 않는다.

데이터 전 처리 및 시각화

3. 데이터 이해를 위한 탐색과 시각화

3.1 Matplotlib

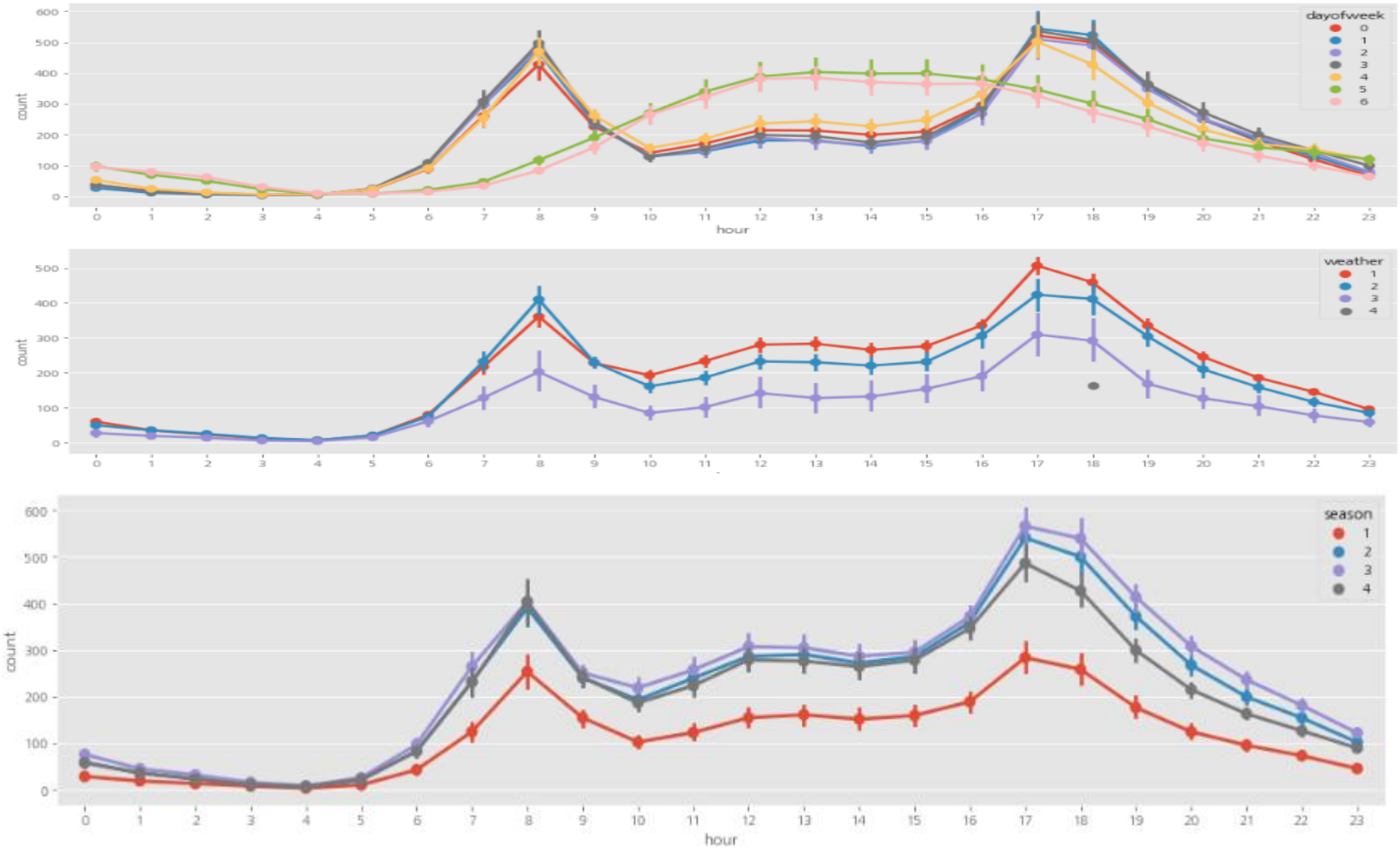
In [69]:
```
fig,(ax1,ax2,ax3,ax4,ax5)= plt.subplots(nrows=5)
fig.set_size_inches(18,25)

sns.pointplot(data=train, x="hour", y="count", ax=ax1)

sns.pointplot(data=train, x="hour", y="count", hue="workingday", ax=ax2)

sns.pointplot(data=train, x="hour", y="count", hue="dayofweek", ax=ax3)

sns.pointplot(data=train, x="hour", y="count", hue="weather", ax=ax4)

sns.pointplot(data=train, x="hour", y="count", hue="season", ax=ax5)
```



데이터 전 처리 및 시각화

3. 데이터 이해를 위한 탐색과 시각화

3.1 Matplotlib



데이터 전 처리 및 시각화

3. 데이터 이해를 위한 탐색과 시각화

3.2 상관계수

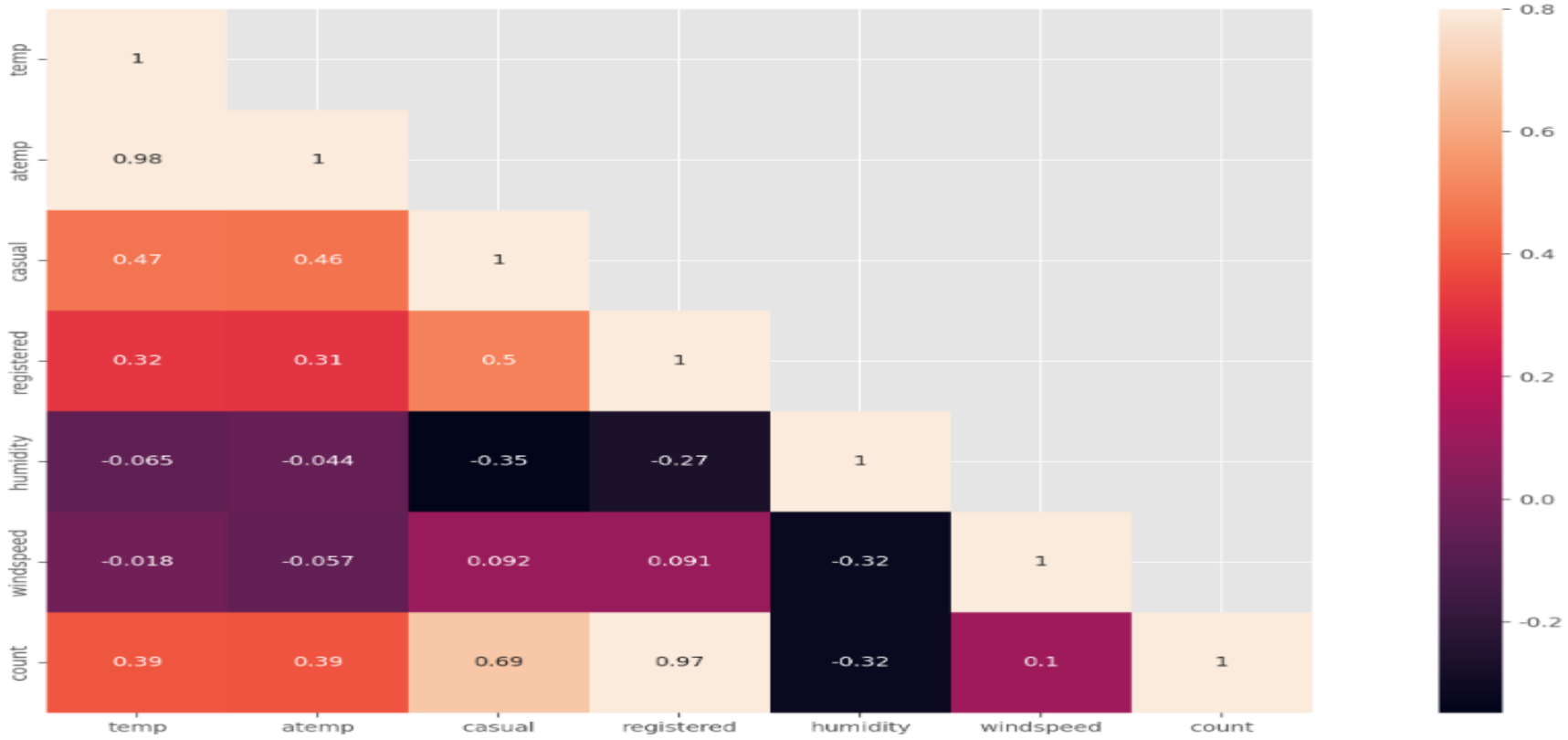
In [70]:
```
corrMatt = train[["temp", "atemp", "casual", "registered", "humidity", "windspeed", "count"]]
corrMatt = corrMatt.corr()
print(corrMatt)

mask = np.array(corrMatt)
mask[np.tril_indices_from(mask)] = False
```

```
                temp     atemp    casual  registered  humidity  windspeed  ₩
temp        1.000000  0.984948  0.467097    0.318571 -0.064949  -0.017852
atemp       0.984948  1.000000  0.462067    0.314635 -0.043536  -0.057473
casual      0.467097  0.462067  1.000000    0.497250 -0.348187   0.092276
registered  0.318571  0.314635  0.497250    1.000000 -0.265458   0.091052
humidity   -0.064949 -0.043536 -0.348187   -0.265458  1.000000  -0.318607
windspeed  -0.017852 -0.057473  0.092276    0.091052 -0.318607   1.000000
count       0.394454  0.389784  0.690414    0.970948 -0.317371   0.101369

               count
temp        0.394454
atemp       0.389784
casual      0.690414
registered  0.970948
humidity   -0.317371
windspeed   0.101369
count       1.000000
```

데이터 전 처리 및 시각화

3. 데이터 이해를 위한 탐색과 시각화

3.2 상관계수



- **온도, 습도, 풍속은 거의 연관관계가 없다.**
- **대여량과 가장 연관이 높은 건 registere로 등록 된 대여자가 많지만, test 데이터에는 이 값이 없다.**
- **Atemp와 temp는 상관관계가 높지만 온도와 체감온도로 피처로 사용하기에 적합하지 않을 수 있다.**

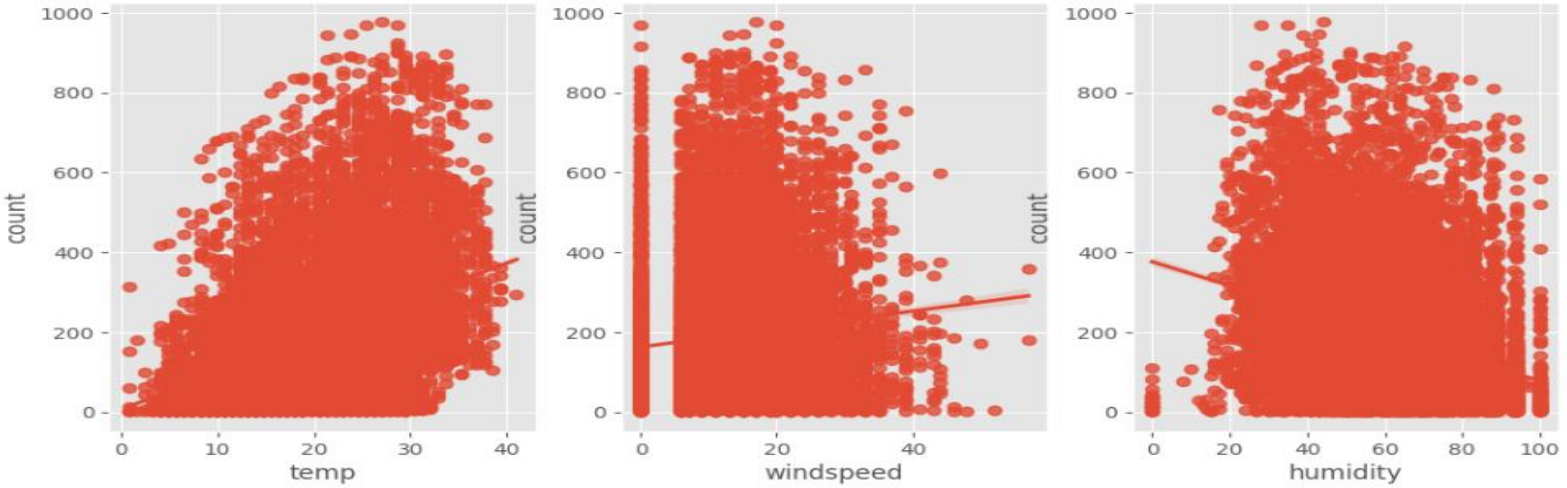
데이터 전 처리 및 시각화

3. 데이터 이해를 위한 탐색과 시각화

3.2 상관계수

In [43]:
```
fig,(ax1,ax2,ax3) = plt.subplots(ncols=3)
fig.set_size_inches(12, 5)
sns.regplot(x="temp", y="count", data=train,ax=ax1)
sns.regplot(x="windspeed", y="count", data=train,ax=ax2)
sns.regplot(x="humidity", y="count", data=train,ax=ax3)
```

Out [43]: <Axes: xlabel='humidity', ylabel='count'>



- **온도, 풍속, 습도에 따른 산점도를 측정해본 결과**
- **풍속인 경우 0에 몰려 있는 data가 많다.**
- **습도도 0과 100에 몰려 있는 data가 많다.**
- **따라서 풍속인 경우 관측되지 않은 수치에 대해 0으로 기록된 것이 아닐까 생각 된다.**

데이터 전 처리 및 시각화

3. 데이터 이해를 위한 탐색과 시각화

3.2 상관계수

In [44]:
```
def concatenate_year_month(datetime):
    return "{0}-{1}".format(datetime.year, datetime.month)

train["year_month"] = train["datetime"].apply(concatenate_year_month)

print(train.shape)
train[["datetime", "year_month"]].head()
```

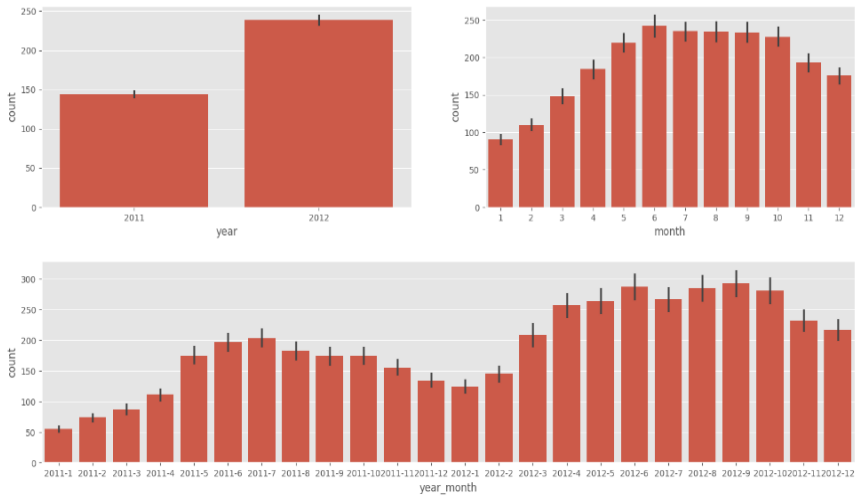
```
(10886, 19)
```

Out[44]:

	datetime	year_month
0	2011-01-01 00:00:00	2011-1
1	2011-01-01 01:00:00	2011-1
2	2011-01-01 02:00:00	2011-1
3	2011-01-01 03:00:00	2011-1
4	2011-01-01 04:00:00	2011-1

In [45]:
```
fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2)
fig.set_size_inches(18, 4)

sns.barplot(data=train, x="year", y="count", ax=ax1)
sns.barplot(data=train, x="month", y="count", ax=ax2)

fig, ax3 = plt.subplots(nrows=1, ncols=1)
fig.set_size_inches(18, 4)

sns.barplot(data=train, x="year_month", y="count", ax=ax3)
```

Out[45]: <Axes: xlabel='year_month', ylabel='count'>



- **2011년보다 2012년의 대여량이 더 많다.**
- **겨울보다는 여름에 대여량이 많으며,**
- **2011년과 2012년의 월별 데이터를 이어보면 전체적으로 증가하는 추세이다.**

데이터 전 처리 및 시각화

4. 데이터 정제

4.1 데이터 정제

In [46]:
```
trainWithoutOutliers = train[np.abs(train["count"] - train["count"].mean()) <= (3*train["count"].std())]

print(train.shape)
print(trainWithoutOutliers.shape)
```

```
(10886, 19)
(10739, 19)
```

- **Outliers를 통하여 data 147개를 삭제**

:
```
# count값의 데이터 분포도를 파악

figure, axes = plt.subplots(ncols=2, nrows=2)
figure.set_size_inches(12, 10)

sns.distplot(train["count"], ax=axes[0][0])
stats.probplot(train["count"], dist='norm', fit=True, plot=axes[0][1])
sns.distplot(np.log(trainWithoutOutliers["count"]), ax=axes[1][0])
stats.probplot(np.log1p(trainWithoutOutliers["count"]), dist='norm', fit=True, plot=axes[1][1])
```

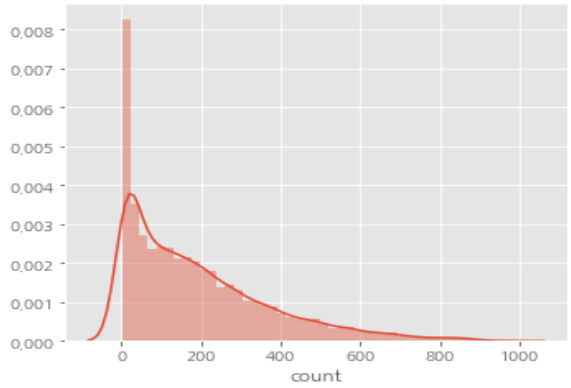
:
```
((array([-3.82819677, -3.60401975, -3.48099008, ...,  3.48099008,
         3.60401975,  3.82819677]),
  array([ 0.69314718,  0.69314718,  0.69314718, ...,  6.5971457 ,
         6.59850903,  6.5998705 ])),
 (1.3486990121229778, 4.5624238680878078, 0.95811767809096149))
```

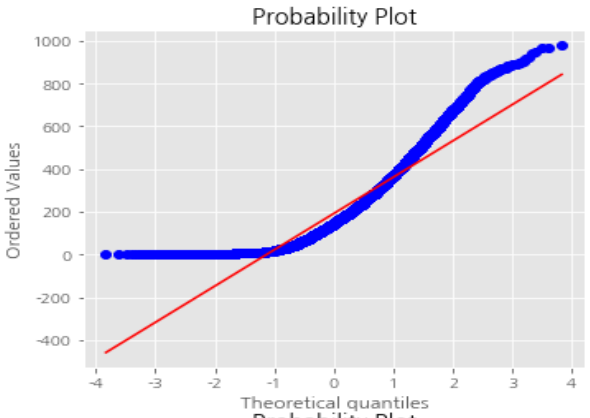
- **Count값의 데이터 분포로를 파악하기 위해 distplot 사용**

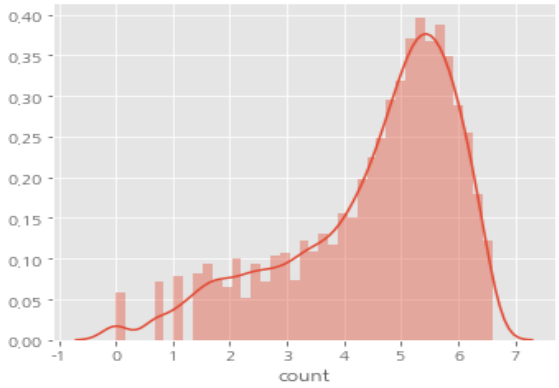
데이터 전 처리 및 시각화

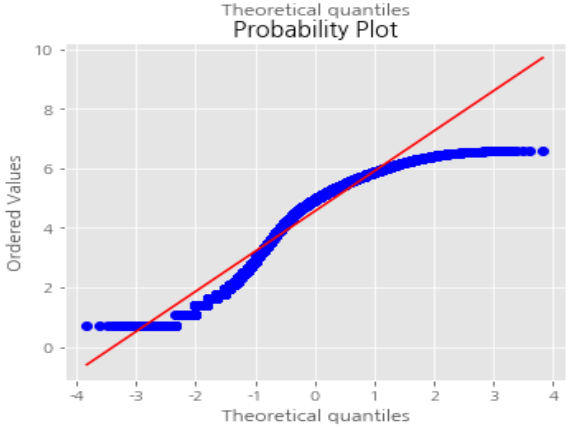
4. 데이터 정제

4.1 데이터 정제









Count변수가 오른쪽에 치우쳐져 있다. 대부분의 기계학습은 종속변수가 normal 이어야 하기에 정규분포를 갖는 것이 바람직하다. 대안으로 outlier data를 제거하고 “count” 변수에 로그를 씌워 변경해 봐도 정규분포를 따르지는 않지만 이전 그래프보다는 좀 더 자세히 표현하고 있다.

데이터 전 처리 및 시각화

6. 특성 스케일링 및 변환

6.1 특성 스케일링 및 변환

```
train = pd.read_csv("data/train.csv", parse_dates=["datetime"])
train.shape
```

```
(10886, 12)
```

```
test = pd.read_csv("data/test.csv", parse_dates=["datetime"])
test.shape
```

```
(6493, 9)
```

train, test data의 datatime을 연, 월, 일, 시, 분, 초, 요일로 나누어 구성

```
train["year"] = train["datetime"].dt.year
train["month"] = train["datetime"].dt.month
train["day"] = train["datetime"].dt.day
train["hour"] = train["datetime"].dt.hour
train["minute"] = train["datetime"].dt.minute
train["second"] = train["datetime"].dt.second
train["dayofweek"] = train["datetime"].dt.dayofweek
train.shape
```

```
(10886, 19)
```

```
test["year"] = test["datetime"].dt.year
test["month"] = test["datetime"].dt.month
test["day"] = test["datetime"].dt.day
test["hour"] = test["datetime"].dt.hour
test["minute"] = test["datetime"].dt.minute
test["second"] = test["datetime"].dt.second
test["dayofweek"] = test["datetime"].dt.dayofweek
test.shape
```

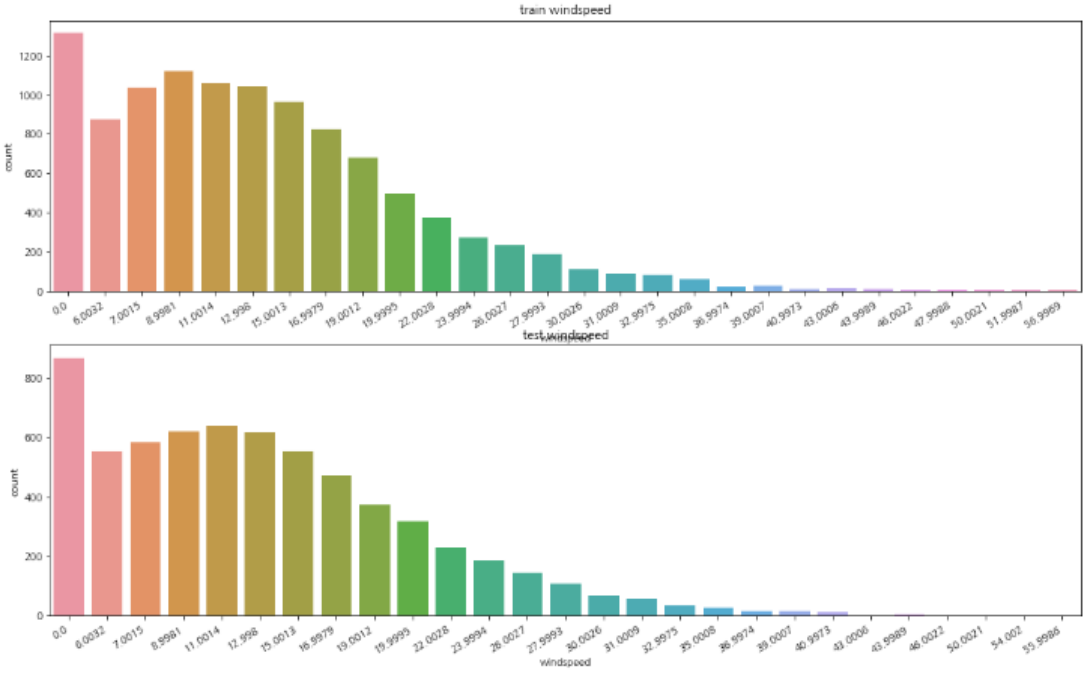
```
(6493, 16)
```

데이터 전 처리 및 시각화

4. 데이터 정제

4.1 데이터 정제

```
# widspeed 풍속에 0 값이 가장 많다. => 잘못 기록된 데이터를 고쳐 줄 필요가 있음
fig, axes = plt.subplots(nrows=2)
fig.set_size_inches(18,10)

plt.sca(axes[0])
plt.xticks(rotation=30, ha='right')
axes[0].set(ylabel='Count',title="train windspeed")
sns.countplot(data=train, x="windspeed", ax=axes[0])

plt.sca(axes[1])
plt.xticks(rotation=30, ha='right')
axes[1].set(ylabel='Count',title="test windspeed")
sns.countplot(data=test, x="windspeed", ax=axes[1])
```



풍속을 시각화해보니 0에 가장 많은 값들이 모여있는것을 확인할 수 있다. 아마도 측정하지 않은 값을 0으로 넣지 않았을까 생각해 본다. 그래서 0인 data들의 값을 보정해주면 예측하는데 도움이 되지 않을까 생각한다.

데이터 전 처리 및 시각화

6. 특성 스케일링 및 변환

6.1 특성 스케일링 및 변환

```
# 그래서 머신러닝으로 예측을 해서 풍속을 넣어주도록 한다.
from sklearn.ensemble import RandomForestClassifier

def predict_windspeed(data):

    # 풍속이 0인것과 아닌 것을 나누어 준다.
    dataWind0 = data.loc[data['windspeed'] == 0]
    dataWindNot0 = data.loc[data['windspeed'] != 0]

    # 풍속을 예측할 피처를 선택한다.
    wCol = ["season", "weather", "humidity", "month", "temp", "year", "atemp"]

    # 풍속이 0이 아닌 데이터들의 타입을 스트링으로 바꿔준다.
    dataWindNot0["windspeed"] = dataWindNot0["windspeed"].astype("str")

    # 랜덤포레스트 분류기를 사용한다.
    rfModel_wind = RandomForestClassifier()

    # wCol에 있는 피처의 값을 바탕으로 풍속을 학습시킨다.
    rfModel_wind.fit(dataWindNot0[wCol], dataWindNot0["windspeed"])

    # 학습한 값을 바탕으로 풍속이 0으로 기록 된 데이터의 풍속을 예측한다.
    windOValues = rfModel_wind.predict(X = dataWind0[wCol])

    # 값을 다 예측 후 비교해 보기 위해
    # 예측한 값을 넣어 줄 데이터 프레임을 새로 만든다.
    predictWind0 = dataWind0
    predictWindNot0 = dataWindNot0

    # 값이 0으로 기록 된 풍속에 대해 예측한 값을 넣어준다.
    predictWind0["windspeed"] = windOValues

    # dataWindNot0 0이 아닌 풍속이 있는 데이터프레임에 예측한 값이 있는 데이터프레임을 합쳐준다.
    data = predictWindNot0.append(predictWind0)

    # 풍속의 데이터타입을 float으로 지정해 준다.
    data["windspeed"] = data["windspeed"].astype("float")

    data.reset_index(inplace=True)
    data.drop('index', inplace=True, axis=1)

    return data
```

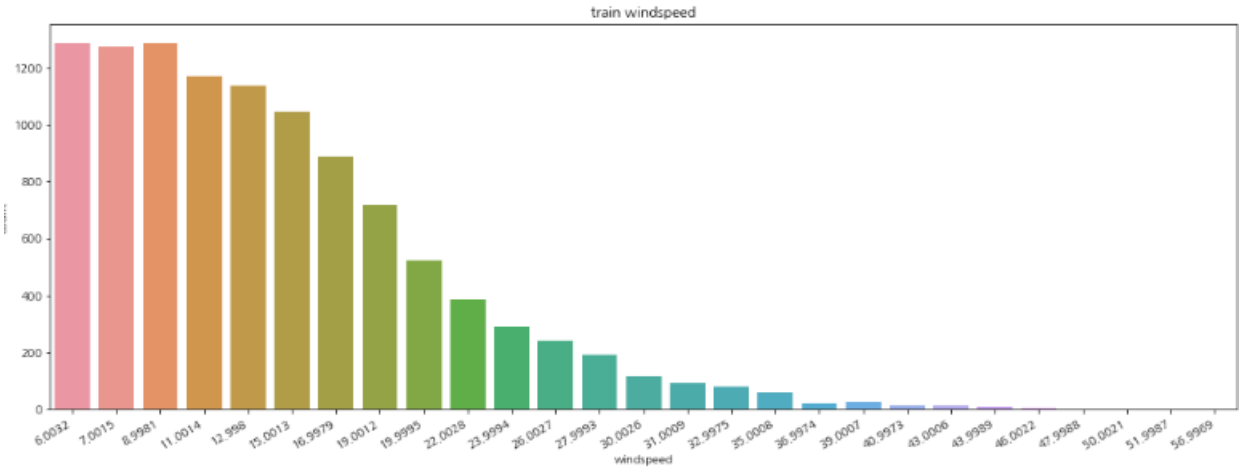
데이터 전 처리 및 시각화

6. 특성 스케일링 및 변환

6.1 특성 스케일링 및 변환

```
# 0값을 조정한다.
train = predict_windspeed(train)
# test = predict_windspeed(test)

# windspeed 의 0값을 조정한 데이터를 시각화
fig, ax1 = plt.subplots()
fig.set_size_inches(18,6)

plt.sca(ax1)
plt.xticks(rotation=30, ha='right')
ax1.set(ylabel='Count',title="train windspeed")
sns.countplot(data=train, x="windspeed", ax=ax1)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x114efdb00>
```



RandomForest로 0인 값들을 예측하여 입력하였더니 0의 값들이 사라지고 예측된 값들로 옮겨진 것을 확인하였다.

데이터 전 처리 및 시각화

6. 특성 스케일링 및 변환

6.1 특성 스케일링 및 변환

```
# 연속형 feature와 범주형 feature
# 연속형 feature = ["temp","humidity","windspeed","atemp"]
# 범주형 feature의 type을 category로 변경 해 준다.
categorical_feature_names = ["season","holiday","workingday","weather",
                             "dayofweek","month","year","hour"]

for var in categorical_feature_names:
    train[var] = train[var].astype("category")
    test[var] = test[var].astype("category")
```

```
feature_names = ["season", "weather", "temp", "atemp", "humidity", "windspeed",
                 "year", "hour", "dayofweek", "holiday", "workingday"]

feature_names
```

```
['season',
 'weather',
 'temp',
 'atemp',
 'humidity',
 'windspeed',
 'year',
 'hour',
 'dayofweek',
 'holiday',
 'workingday']
```

- 연속형 feature와 범주형 feature가 있는데 범주형 feature인 경우 값이 두배가 된다고 하여 두배로 증가하지 않는 범주를 말하며 범주형 feature는 one hot encodin을 통하여 머신러닝이 좀 더 쉽게 이해할 수 있도록 인코딩을 한다.

- feature 선택은 위와 같이 season, weather, temp, atemp, humidity, windspped, year, hour, dayofweek, holiday, workingday를 선택하였다.

데이터 전 처리 및 시각화

6. 특성 스케일링 및 변환

6.1 특성 스케일링 및 변환

```
X_train = train[feature_names]

print(X_train.shape)
X_train.head()
```

(10886, 11)

	season	weather	temp	atemp	humidity	windspeed	year	hour	dayofweek	holiday	workingday
0	1	2	9.84	12.880	75	6.0032	2011	5	5	0	0
1	1	1	15.58	19.695	76	16.9979	2011	10	5	0	0
2	1	1	14.76	16.665	81	19.0012	2011	11	5	0	0
3	1	1	17.22	21.210	77	19.0012	2011	12	5	0	0
4	1	2	18.86	22.725	72	19.9995	2011	13	5	0	0

```
X_test = test[feature_names]

print(X_test.shape)
X_test.head()
```

(6493, 11)

	season	weather	temp	atemp	humidity	windspeed	year	hour	dayofweek	holiday	workingday
0	1	1	10.66	11.365	56	26.0027	2011	0	3	0	1
1	1	1	10.66	13.635	56	0.0000	2011	1	3	0	1
2	1	1	10.66	13.635	56	0.0000	2011	2	3	0	1
3	1	1	10.66	12.880	56	11.0014	2011	3	3	0	1
4	1	1	10.66	12.880	56	11.0014	2011	4	3	0	1

train, test data에 선택해준 feature로 새로운 행렬을 만든다.

데이터 전 처리 및 시각화

6. 특성 스케일링 및 변환

6.1 특성 스케일링 및 변환

```
label_name = "count"

y_train = train[label_name]

print(y_train.shape)
y_train.head()
```

```
(10886,)
0     1
1    36
2    56
3    84
4    94
Name: count, dtype: int64
```

label될 data는 train data에 있는 count값이 될 것이다. 이 값을 바탕으로 test의 count를 예측할 예정이다.

모델 선택 및 훈련

모델 선택 및 훈련

```
from sklearn.ensemble import RandomForestRegressor

max_depth_list = []

model = RandomForestRegressor(n_estimators=500,
                              n_jobs=-1,
                              random_state=0)
model
```

RandomForestRegressor

RandomForestRegressor(n_estimators=500, n_jobs=-1, random_state=0)

```
%time score = cross_val_score(model, X_train, y_train, cv=k_fold, scoring=rmsle_scorer)
score = score.mean()
# 0에 근접할수록 좋은 데이터
print("Score= {0:.5f}".format(score))
```

```
CPU times: total: 12.9 s
Wall time: 24.2 s
Score= 0.32932
```

```
# 학습시킴, 피팅(옷을 맞출 때 사용하는 피팅을 생각함) - 피처와 레이블을 넣어주면 알아서 학습을 함
model.fit(X_train, y_train)
```

RandomForestRegressor

RandomForestRegressor(n_estimators=500, n_jobs=-1, random_state=0)

모델 선택 및 훈련

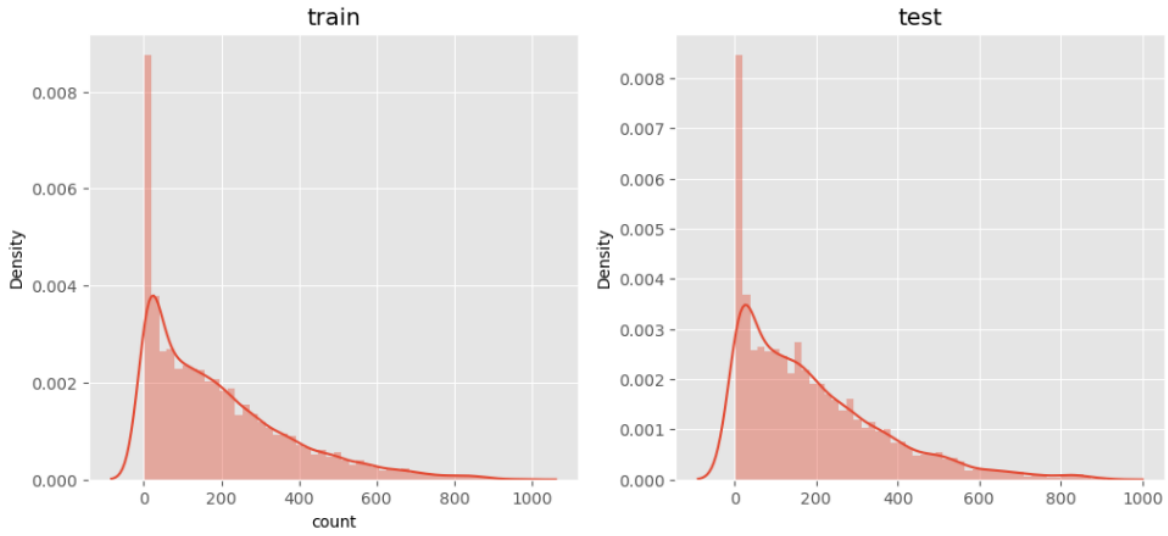
모델 선택 및 훈련

```
# 예측
predictions = model.predict(X_test)

print(predictions.shape)
predictions[0:10]
```

```
(6493,)

array([ 12.14 ,   4.86 ,   3.974,   3.494,   2.992,   6.618,  38.404,
       106.92 , 239.758, 136.894])
```

```
# 예측한 데이터를 시각화 해본다.
fig,(ax1,ax2)= plt.subplots(ncols=2)
fig.set_size_inches(12,5)
sns.distplot(y_train,ax=ax1,bins=50)
ax1.set(title="train")
sns.distplot(predictions,ax=ax2,bins=50)
ax2.set(title="test")
```

```
[Text(0.5, 1.0, 'test')]
```



모델 선택 및 훈련

모델 선택 및 훈련

```
from sklearn.ensemble import RandomForestRegressor

max_depth_list = []

model = RandomForestRegressor(n_estimators=500,
                              n_jobs=-1,
                              random_state=0)
model
```

RandomForestRegressor

RandomForestRegressor(n_estimators=500, n_jobs=-1, random_state=0)

```
%time score = cross_val_score(model, X_train, y_train, cv=k_fold, scoring=rmsle_scorer)
score = score.mean()
# 0에 근접할수록 좋은 데이터
print("Score= {0:.5f}".format(score))
```

```
CPU times: total: 12.9 s
Wall time: 24.2 s
Score= 0.32932
```

```
# 학습시킴, 피팅(옷을 맞출 때 사용하는 피팅을 생각함) - 피처와 레이블을 넣어주면 알아서 학습을 함
model.fit(X_train, y_train)
```

RandomForestRegressor

RandomForestRegressor(n_estimators=500, n_jobs=-1, random_state=0)

모델 선택 및 훈련

모델 선택 및 훈련

```
submission = pd.read_csv("data/sampleSubmission.csv")
submission

submission["count"] = predictions

print(submission.shape)
submission.head()
```

```
(6493, 2)
```

	datetime	count
0	2011-01-20 00:00:00	12.140
1	2011-01-20 01:00:00	4.860
2	2011-01-20 02:00:00	3.974
3	2011-01-20 03:00:00	3.494
4	2011-01-20 04:00:00	2.992

```
submission.to_csv("data/Score_{0:.5f}_submission.csv".format(score), index=False)
```

“인류의 생명과 건강”

끊임없는 연구 개발 및 차세대 신기술 개발을 위한 투자,

세계 무대에서의 노하우 및 단단한 Partnership을 통해

첨단생명 공학 전문회사로 한 단계 더 성장하여

인류의 생명과 건강을 책임지겠습니다.

META BIOMED CO., LTD.

End of document